Internal Assessment Test 2– AUGUST 2022

Solutions for IAT2_PSP

1 (a) Compare cascade if-else with switch statement

1. In the else *if ladder*, the control goes through the every else if statement until it finds the true value of the statement or it comes to the end of the else if ladder. In the *switch case*, as per the value of the switch, the control jumps to the corresponding case.
2. The switch case is more compact than a lot of nested else if. So, switch is considered to be more readable.
3. The use of break statements in switch is essential but there is no need of use of break in else if ladder.
4. The variable data type that can be used in expression of switch is integer only whereas in else if ladder accepts integer type as well as character.
5. Since the compiler is capable of optimizing the switch statement, they are generally considered to be more efficient. Each case in the switch statement is independent of the previous one. In case of else-if ladder, the code needs to be processed in the order determined by the programmer.
6. *Switch case statement* work on the basis of equality operator whereas *else if ladder* works on the basis of true false(zero/non-zero) basis.

1 (b) Write a C program for simple calculator for two integers using switch statement

```
#include <stdio.h>
int main(){
   char ch;
   int a, b, result;

   // Asking for Input
   printf("Enter an Operator (+, *, *, /): ");
   scanf("%c", &ch);
   printf("Enter two operands: \n");
   scanf("%d %d", &a, &b);
   switch(ch)
{
      case '+':
         result = a + b;
         break;
      case '-':
         result = a - b;
         break;
          case '*':
         result = a * b;
         break;
      case '/':
       if(b==0)
```

```
        {
        printf("Division by zero error\n");
        }
        else
        {
                result = a / b;
         }
        break;
default:
printf("\n Enter correct option\n");
break;
}
   printf("Result = %d", result);
   return 0;
}
```

Output:

2(a) Explain declaration and initialization of 1-D array with example

Single Dimensional Array(1-D): An array with one index is called as Single dimensional array (One-dimensional array)

- **Declaration of one-dimensional array:** As we declare the variables before they are used in a program, an array must also be declared and defined before it is used using following syntax
  - syntax:

    datatype arrayname[size];

    where, **data_type:** data type can be int, float, char etc.

    array_name: name of the array which is a valid C variable.

    size: it is the number of elements in the array.
- **Examples:**

  float height[5]; int marks[5]; char name[5];
- **5 memory locations are reserved. sizeof(int) is 2 bytes, 2*5=10 bytes are reserved.**
- **5 memory locations are reserved. sizeof(char) is 1 bytes 1*5=5 bytes are reserved.**

Initialization of one-dimensional array: "Process of assigning the values to the individual elements of an array is called as Initialization of an array".

Syntax:

data_type array_name[size]={v1,v2,-----,vn};

where, v1,v2,-----,vn are the values and should be enclosed within '{' and '}' separated by commas.

Examples:

```
int marks[5] = {76,45,-12,87,92};
int a[5] = {0};
/* if one element is initialized, rest of the elements will be initialized to zero */
int a[5] = {1};
/* first element a[0] is  = 1, rest all will be initialized with 0 */
```

2(b) Write a C Program to find largest and smallest in array of N integers
/ C program to find the smallest and largest element in an array

```
#include<stdio.h>

int main()
{
int a[50],i,n,large,small;
printf("\nEnter the number of elements : ");
scanf("%d",&n);
printf("\nInput the array elements : ");
for(i=0;i<n;++i)
scanf("%d",&a[i]);

large=small=a[0];

for(i=1;i<n;++i)
{
if(a[i]>large)
large=a[i];

if(a[i]<small)
small=a[i];
}

printf("\nThe smallest element is %d\n",small);
printf("\nThe largest element is %d\n",large);

return 0;
}
```

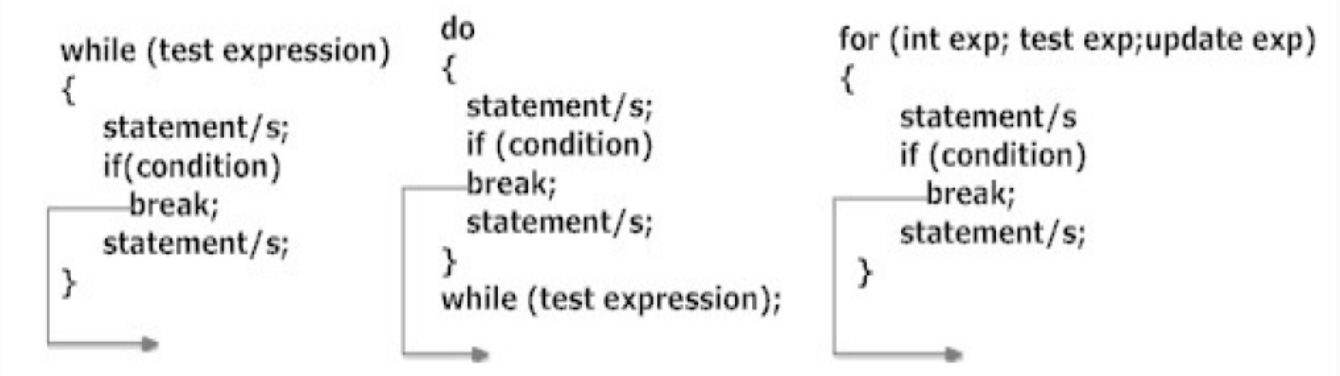
OUTPUT:

```
Enter the number of elements : 5
Input the array elements : 1 2 3 4 5
The smallest element is 1
The largest element is 5
```

3(a) Demonstrate the working of break and continue statement with a suitable example.

Break:

- A break statement can be used to terminate or to come out from the loop or conditional statement unconditionally.
- It can be used in switch statement to break and come out from the switch statement after each case expression




```
int main()
{
   int cnt = 0;
```

```
    do
    {
        printf("Value: %d", cnt);
        cnt++;
        if(cnt>5)
        {
            break;   //terminate the loop
        }
    }
    while(cnt<10);
    return 0;
}
```

Continue: A continue statement can be used into the loop when we want to skip some statement to be executed and continue the execution of the above statement based on some specific condition.

- ❖ Similar to break statement, continue is also used with if statement.
- ❖ In For Loop, the continue statement causes the conditional test and increment/ decrement statement of the loop to be executed.
- ❖ In While Loop, the continue statement takes control of the condition statement.
- ❖ In Do-While Loop, continue statement takes control of the condition statement specified in the while loop.

```
int main()
{
    int cnt = 0;
    do
    {
        cnt++;
        if(cnt>5 && cnt<10)
            continue;
             printf("Value: %d", cnt);
    }
    while(cnt<11);
    return 0;
}
```

3(b) Write a C program to plot Pascal's triangle in 11 rows

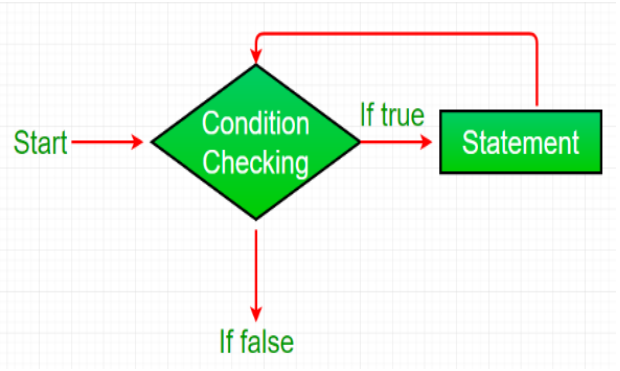
```
#include <stdio.h>
int main()
{
 int i,j,c;
 printf("The Pascal's triangle\n");
 for(i = 0; i<11; i++)
  {
   for(j = 1; j<20-i; j++)
     printf("  "); /* to print two blanks */

   for(j=0; j<i; j++)
   {
     if(i==0 || j==0)
           c=1;
     else
           c=c*(i-j)/j;
     printf("%4d",c);
   }
   printf("\n");
```

	} return (0); } Output: The Pascal's triangle 1 1 1 1 2 1 1 3 3 1 1 4 6 4 1 1 5 10 10 5 1 1 6 15 20 15 6 1 1 7 21 35 35 21 7 1 1 8 28 56 70 56 28 8 1 1 9 36 84 126 126 84 36 9 1

4(a)	Write a C program to implement Selection Sort technique (ascending order). Trace the steps for the input: 9, 1, 6, 3, 2, 0, 5
	Program:
	#include<stdio.h>
	void main( )
	{
	int a[100],i,j,n,temp,pos;
	printf("Enter the value of n\n");
	scanf("%d",&n);
	printf("Enter the numbers in unsorted order:\n");
	for(i=0;i<n;i++)
	scanf("%d", &a[i]);
	for(i=0;i<n;i++)
	{
	pos=i;
	for(j=i+1;j<n;j++)
	{
	if(a[pos]>a[j])
	pos=j;
	}
	if(pos !=i)
	{
	temp=a[i];
	a[i]=a[pos];
	a[pos]=temp;
	}
	}
	printf("The sorted array is\n");
	for(i=0;i<n;i++)
	printf("%d\n",a[i]);
	}
	
	output:
	Enter the value of n
	6
	Enter the numbers in unsorted order:

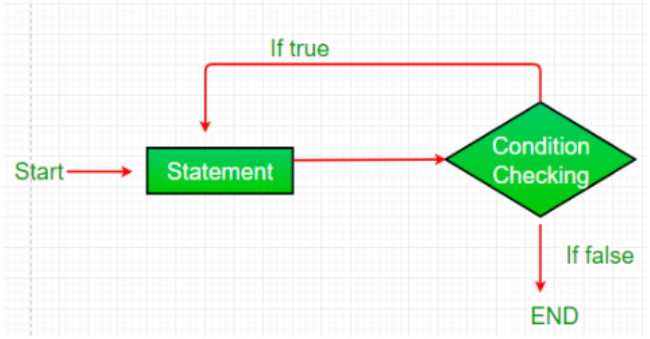
	9 1 6 3 2 0 The sorted array is 0 1 2 3 6 9 Tracing: 1. At present, **9** is stored at the first position, after searching the entire array, it is found that **0** is the smallest value. 2. So, swap 9 with 0. After the first iteration, 0 will appear at the first position in the sorted array. 3. For the second position, where 1 is stored presently, we again sequentially scan the rest of the items of the unsorted array. After scanning, we find that 1 i.e the second element is the lowest element in the array that should appear at second position. 4. So, after two iterations, the two smallest values are placed at the beginning in a sorted way. 5. The same process is applied to the rest of the array elements. Now, we are showing a pictorial representation of the entire sorting process. 6. The sorted array is 0,1,2,3,6,9

5(a)	Compare the working of for, while and do..while along with their syntax and flowchart.
	While Loop:
	❖ It is an entry-controlled loop. In a while loop, a condition is evaluated before processing a body of the loop. If a condition is true then and only then the body of a loop is executed.
	❖ After the body of a loop is executed then control again goes back at the beginning, and the condition is checked if it is true, the same process is executed until the condition becomes false.
	❖ Once the condition becomes false, the control goes out of the loop.
	Syntax :
	while (boolean condition)
	{
	loop statements...
	}
	Example:#include<stdio.h>
	int main()
	{
	int i = 1;
	
	// while loop from 1 to 5
	while (i <= 5)
	{
	printf("%d\n", i);
	i++;
	}
	return 0;
	}
	
	 
	
	Do-While:
	❖ A do-while loop is similar to the while loop except that the condition is always executed after the body of a loop.
	❖ The while loop construct as we discussed in the previous section, makes a test of condition before the loop is executed.

❖ Therefore, the body of the loop may not be executed at all if the condition is not satisfied at the very first attempt. do-while is also called an exit-controlled loop

Syntax:

```
do
{
   statements..
}
while (condition);
```



Example:

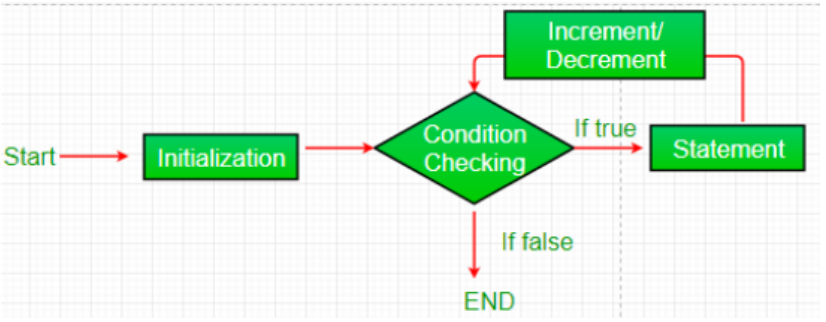
```
#include<stdio.h>
int main()
 {
   int i = 1;
   // do...while loop from 1 to 5
   do {
      printf("%d\n", i);
      ++i;
   }
   while (i <= 5);
   return 0;
}
```

For Loop:

❖ for loop provides a concise way of writing the loop structure.
❖ The initial value of the for loop is performed only once.
❖ The test-condition is a Boolean expression that tests and compares the counter to a fixed value after each iteration, stopping the for loop when false is returned.
❖ The incrementation/decrementation increases (or decreases) the counter by a set value

Syntax:

```
for (initialization condition; testing condition; increment/decrement)
{
   statement(s)
}
```



Examples:

```
#include<stdio.h>
int main()
{
      for (int i = 1; i <= 5; ++i)
{
 printf("%d\n", i);
   }
   return 0;
}
```

6(a)	How are actual parameters different from formal parameters? Explain

Actual or Real Parameters

❖ The variables that are used when a function is invoked are called actual parameters.
❖ Actual parameters are used in the Calling function when a function is invoked.
❖ Actual parameters send values or addresses to the formal parameters. Formal parameters receive them and use the same values.
❖ Actual parameters can be constants, variables or expressions.
  ■ Ex: res = add (m, n);

Formal or Dummy Parameters

❖ The variables defined in the function header or function definition are called formal parameters.
❖ All the variables should be separately declared and each declaration must be separated by

commas.

- ❖ The formal parameters receive values from the actual parameters.
- ❖ If the formal parameters receive the address from the actual parameters, then they should be declared as pointers.
- ❖ The formal parameters should be only variables. Expressions and constants are not allowed.
  - ■ Ex: int add(int a,int b);

6(b) Write a C program to put even and odd elements of an array in two separate arrays. Take the size of the array and array elements as user input.

```
#include<stdio.h>
int main()
{
  int n, a[20];
  printf("Enter the size of the array: ");
  scanf("%d", &n);
  printf("Enter array elements: \n");
  for(int i=0; i<n; i++)
  {
    scanf("%d",&a[i]);
  }
  printf("Even numbers in the array are: \n");
  for(int i=0; i<n; i++)
  {
   if(a[i]%2==0)
   printf("%d ", a[i]);
  }
  printf("\nOdd numbers in the array are: \n");
  for(int i=0; i<n; i++)
  {
   if(a[i]%2!=0)
   printf("%d ", a[i]);
  }

  return 0;
}
```

Output: Enter the size of the array: 5
Enter array elements:
10
15
20
54
26
Even numbers in the array are:
10 20 54 26
Odd numbers in the array are:
15

7(a) Discuss the implementation of user-defined function with a suitable example

- ❖ The functions written by the programmer/user to do the specific tasks are called User-Defined/ Programmer Defined Functions.
- ❖ Elements of User-Defined Functions
- ❖ The three elements of user-defined functions are shown below:
  - ➢ i. Function Prototype/Declaration
  - ➢ ii. Function Definition

➢ iii. Function Call

Function Prototype/Declaration

- As we normally declare the variables before they are used, the functions also should be declared before they are used.
- The process of declaring the functions before they are used (or called) in the program is called Function Prototype.
- The function prototype is also called a Function declaration.
- It does not contain the body of the function.
  - Syntax: **return_type function_name(parameter list);**

Function Definition

- The program module that is written to achieve a specific task is called function definition.
- Each function definition consists of two parts:
  - Function Header
  - Function Body
    - Syntax: return_type function_name (parameter list)

```
{
        declaration part;
        executable part;
        return statement;
}
```

Function Call

- Once the function is defined, it has to be called so as to achieve the task. This method of calling a function to achieve a specified task is called Function Call.
- The function can be called by writing the name of the function and passing the appropriate number of arguments.
- The number of arguments in the function call and number of parameters in the function definition must match.
- Also the order of arguments in the function call and parameters in the function definition must match.
  - Example: add(m,n);

7(b) Write a C program to check a string (with blanks) whether it is palindrome

```
#include <stdio.h>
#include <string.h>
int main(){
   char string1[20];
   int i, length;
   int flag = 0;
   printf("Enter a string:");
   scanf("%s", string1);
   length = strlen(string1);
   for(i=0;i < length ;i++){
      if(string1[i] != string1[length-i-1]){
         flag = 1;
         break;
        }
      }
   if (flag) {
      printf("%s is not a palindrome", string1);
   }
   else {
      printf("%s is a palindrome", string1);
   }
```

```
    return 0;
}
```

8(a) Explain 5 string handling functions without using the built-in function.

```
// STRING LENGTH WITHOUT USING BUILTIN FUNCTION
#include<stdio.h>
#include<string.h>
void main()
{
char str[20];
int length=0;
printf("Enter the string\n");
gets(str);
while(str[length]!='\0')
{
   length ++;
}
printf("Length of the string is=%d\n", length);
}

//STRING COPY
#include<stdio.h>
#include<string.h>
void main()
{
char str1[100],str2[100];
int i;
printf("Enter the string\n");
gets(str1);
for(i=0; str1[i]!='\0'; i++)
{
str2[i] = str1[i];
}
str2[i]='\0';
printf(" string 1 is  is=%s\n", str1);
  printf(" string  2 is is=%s\n", str2);
}

//STRING CONCATENATE
#include<stdio.h>
#include<string.h>
void main()
{
char s1[15]= “Good”;
char s2[15]= “Morning”, s3[20];
int k,i;
```

```
for (i=0; s1[i]!='\0'; i++)
{
s3[i] = s1[i];
k = k +1;
}
for (i=0; s2[i]!='\0'; i++)
{
s3[i] = s2[i];
k = k +1;
}
s3[k] = '\0';
printf("The concatenated String=%s",s3);
}
//STRING COMPARE
#include<stdio.h>
#include<string.h>
void main()
{
  int i;
  char s1[10]="Hello";
char s2[10]="Hey";
len1 = strlen(s1);
len2 = strlen(s2);
if (strlen(s1) != strlen(s2))
printf("The two strings are different");
else{
for (i=0; s1[i]!='\0'; i++)
{
if(s1[i]!=s2[i])
printf("strings are different");
break;
}
}
printf("The two strings are identical");
}
//Convert String to Uppercase without using strupr()

#include <stdio.h>
#include <string.h>
int main()
{
        char Str1[100];
        int i;
        printf("\n Please Enter a String that you want to Convert into Uppercase :  ");
        gets(Str1);
        for (i = 0; Str1[i]!='\0'; i++)
        {
                if(Str1[i] >= 'a' && Str1[i] <= 'z')
```

```
                {
                        Str1[i] = Str1[i] -32;
                }
        }
        printf("\n The given String in Upper Case = %s", Str1);
        return 0;
}
```